GROUP : CEN1.2B 1st Year NAME : STAN OANA-ADRIANA

ALGORITHM DESIGN HOMEWORK
-TECHINCAL REPORT-

1. HOMEWORK assignment problem description

A fisherman is exploring a coastal region rich in lobsters, each with its own size (in centimeters) and value (in gold coins). The fisher-man's net has a limited capacity, expressed in the total number of large and very large data sets randomly generated centimeters it can hold. Given a detailed list of the sizes and values of the lobsters available in that region, your task is to develop a strategy for the fisherman to select lobsters in such a way as to maximize the total value of his catch while adhering to the net's capacity limit. You need to decide which lobsters to include in the net and which to leave behind so that the sum of the values of the selected lobsters is as high as possible, without the sum of their sizes exceeding the capacity of the net. Imagine a scenario where a fisherman is given the opportunity to choose from a selection of lobsters, each with a specified size and value, to fill his net which has a maximum capacity. The fisherman's goal is to maximize the total value of his catch without exceeding the net's size limit.

Here's an example:

Lobsters available:

• Lobster A: Size = 4 cm, Value = 20 gold coins

• Lobster B: Size = 3 cm, Value = 15 gold coins

• Lobster C: Size = 2 cm, Value = 10 gold coins

• Lobster D: Size = 5 cm, Value = 25 gold coins

Net capacity: 10 cm

The challenge is to select the combination of lobsters that maximizes the total value without exceeding a total size of 10 cm.

One possible solution would involve choosing Lobsters A and C, giving us a total size of 6 cm (4 cm + 2 cm) and a total value of 30 gold coins (20 + 10). However, a better solution would be to choose:

Lobsters B, C, and D, which together have a total size of 10 cm (3 cm + 2 cm + 5 cm) and offer a higher total value of 50 gold coins (15 + 10 + 25). This combination exactly fills the net's capacity and maximizes the catch's value.

2. Pseuodocode algorithm :

```
Function knapsack(sizes, values, n, capacity) do
    // Allocate memory dynamically for the dp matrix
    dp ← Create a (n + 1) x (capacity + 1) matrix
    // Initialize the dp table with 0
    For i from 0 to n do
        For w from 0 to capacity do
            If i == 0 OR w == 0 then
                dp[i][w] ← 0
            Else If sizes[i - 1] <= w then
                // We have two choices:
                // 1. Exclude the current item and take the maximum value obtained by the previous items with the same capacity
                // 2. Include the current item and add its value to the maximum value obtained by the previous items with the reduced capacity
                dp[i][w] ← max(dp[i - 1][w], dp[i - 1][w - sizes[i - 1]] + values[i - 1])
            Else
                // If the size of the current item is greater than the current capacity, exclude the current item
                dp[i][w] ← dp[i - 1][w]
            End If
        End For
    End For
    // Compute the final result
    result ← dp[n][capacity]
    // Free allocated memory
    For i from 0 to n do
        Free(dp[i])
    End For
    Free(dp)
    Return result
End Function
```

3. EXPERIMENTAL DATA

This section presents the experimental data used in the analysis. The data is divided into 10 tests, each containing 5 lobsters represented by a size-value pair. The capacity of the fishing net for each test is also provided.

Number of tests : 10

Number of lobsters per test : 5

***First row of each testcase is represented by : fishing net capacity and maximum value range.**

```
7  10
2 8
5 1
10 5
9 9
3 5

125  100
6 46
2 28
2 92
6 43
8 37
```

```
165  300
2 205
3 154
3 83
2 117
9 96

376  500
8 227
2 39
10 413
8 300
6 395

568  700
4 12
3 234
4 465
2 12
4 569

689  900
8 645
3 358
8 260
4 742
10 779

890  1100
7 836
1 743
9 407
1 143
5 649
```

```
3467  10000
7 3806
1 6730
1 5351
7 1102
4 3549

45678  100000
10 12624
5 19955
7 11841
7 7377
2 26309

678960  1000000
5 32440
7 11324
8 21539
9 2083
10 16542
```

4. Results & Conclusions

I porvided in this section all the results and time complexities for each testcase.

Test case 1:

Maximum value of lobsters that can be caught: 13

Time Complexity: 0.000000

Test case 2:

Maximum value of lobsters that can be caught: 246

Time Complexity: 0.000000

Test case 3:

Maximum value of lobsters that can be caught: 655

Time Complexity: 0.000000

Test case 4:

Maximum value of lobsters that can be caught: 1374

Time Complexity: 0.000000

Test case 5:

Maximum value of lobsters that can be caught: 1292

Time Complexity: 0.000000

Test case 6:

Maximum value of lobsters that can be caught: 2784

Time Complexity: 0.000000

Test case 7:

Maximum value of lobsters that can be caught: 2778

Time Complexity: 0.000000

Test case 8:

Maximum value of lobsters that can be caught: 20538

Time Complexity: 0.000000

Test case 9:

Maximum value of lobsters that can be caught: 78106

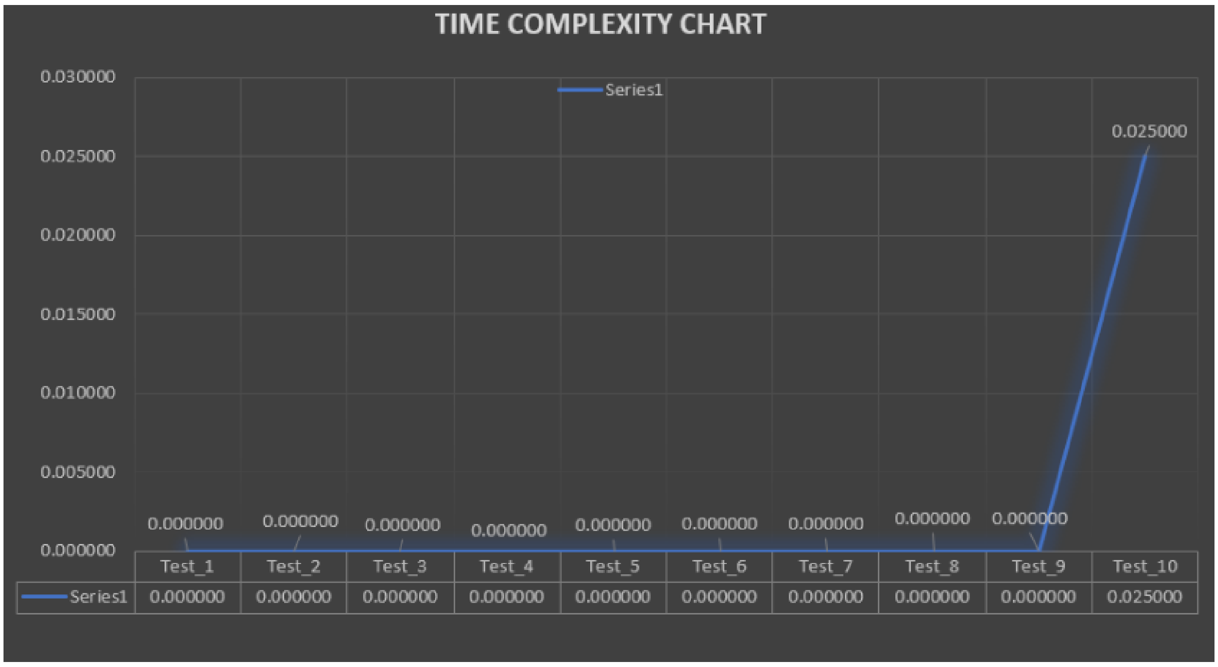
Time Complexity: 0.000000

Test case 10:

Maximum value of lobsters that can be caught: 83928

Time Complexity: 0.025000

The results presented here are the expected ones and the time complexities are also correct. Now, I am gonna present you the chart with the time complexities:



	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_10
Series1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.025000

Conclusions :

- **For the first 9 testcases we have the cpu time complexity of 0 seconds because C has the power to compute the numbers up to 10.000 very quickly. After that bound, as you could see in the 10th case it take 0.025 seconds which is also very well.**
- **So, as expected, the chart is increasing.**

Github link : Stan-Oana/TEMA_DE_CASA_AD (github.com)